VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

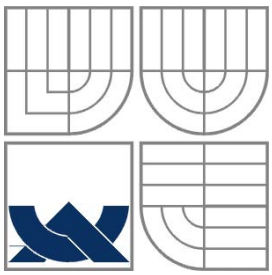
LOGICKÁ HRA FUTOSHIKI

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

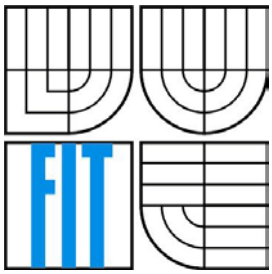
AUTOR PRÁCE
AUTHOR

TOMÁŠ BENEDIKTI

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

LOGICKÁ HRA FUTOSHIKI

FUTOSHIKI PUZZLE

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — TOMÁŠ BENEDIKTI
AUTHOR

VEDOUCÍ PRÁCE — ING. EVA ZÁMEČNÍKOVÁ
SUPERVISOR

BRNO 2010

Abstrakt

Tato práce se zaobírá problematikou logických her a hlavolamů. Hlavním zaměřením je hra Futoshiki, její historie a pravidla. Detailně se popisují algoritmy použité ve vytvořené aplikaci, od jejich návrhu po implementaci. Poslední částí je tvorba uživatelského rozhraní a testovaní úspěšnosti aplikace.

Abstract

This thesis deals with the topic of logical games and puzzles. The main focus is on the Futoshiki puzzle, its history and rules. The algorithms created for the application are covered in depth, from the design to the implementation. The last part contains the creation of the user interface and testing the success of the application.

Klíčová slova

futoshiki, logika, hra, hlavolam, GUI, aplikace

Keywords

futoshiki, logic, game, puzzle, GUI, application

Citace

Benedikti Tomáš: Logická hra Futoshiki, bakalářská práce, Brno, FIT VUT v Brně, 2010

Logická hra Futoshiki

Prohlášení

Prohlašuji, že jsem tuhle bakalářskou práci vypracoval samostatně pod vedením Ing. Evy Zámečníkové. Všechny literární prameny a publikace, ze kterých jsem čerpal, jsou uvedeny v seznamu použité literatury.

……
Tomáš Benedikti
22.01.2010

Poděkování

Chcel by som sa poďakovať Ing. Eve Zámečníkovej za jej rady a čas pri tvorbe tejto bakalárskej práce. Hlavne za jej ochotu rýchle reagovať na otázky a pomoc pri riešení niektorých problémov.

Obsah

Obsah 1
1 Úvod 2
2 Problematika logických hier 3
2.1 Nástup logických hier 3
2.2 Sudoku 3
2.3 Ďalšie logické hry 3
3 Hra Futoshiki 5
3.1 História 5
3.2 Pravidlá 5
3.3 Postup riešenia 5
3.4 Dekompozícia tvorby aplikácie 6
4 Algoritmus tvorby zadania hry 7
4.1 Návrh 7
4.2 Implementácia 7
4.3 Výhody a nevýhody riešenia 9
5 Algoritmus pre kontrolu riešenia hry 10
5.1 Návrh 10
5.2 Implementácia 10
5.3 Výhody a nevýhody riešenia 10
6 Algoritmus pre riešenie hry 11
6.1 Návrh 11
6.2 Implementácia 11
6.3 Výhody a nevýhody riešenia 13
7 Užívateľské prostredie hry 14
7.1 Návrh 14
7.2 Proces tvorby 14
7.3 Finálna implementácia 15
7.4 Úspešnosť aplikácie 18
8 Záver 19
Literatúra 20

1 Úvod

Táto bakalárska práca sa bude zaoberať problematikou logických hier a popisom ich riešenia. Hlavný dôraz bude kladený na hru Futoshiki, ktorá vznikla ako odnož hry Sudoku. Jednotlivé kapitoly sa zaoberajú pravidlami logických hier, algoritmami tvorby a riešenia hry Futoshiki. Okrem toho sa tu bude nachádzať aj vysvetlenie správnej tvorby užívateľského rozhrania a samotná tvorba rozhrania aplikácie Futoshiki. Rozdelenie a obsah častí bude teda nasledovné:

- začneme úvodom do problematiky logických hier, vymenovaním niektorých a spomenutím ich pravidiel
- ďalšia časť už bude pojednávať konkrétne o hre Futoshiki - o jej vzniku, pravidlách, riešení a nakoniec o rozdelení úloh pri tvorbe aplikácie, ktorá bude generovať/riešiť túto hru
- postup tvorenia riešiteľného zadania tejto hry sa nachádza vo štvrtej kapitole
- nasleduje kratšia kapitola zaoberajúca sa vyriešením užívateľovho riešenia hry
- šiesta kapitola obsahuje návrh a implementáciu algoritmu riešenia hry Futoshiki
- na záver sú zhrnuté poznatky z tvorby, postrehy z fungovania logickej hry a efektívnosť/správnosť navrhnutého riešenia

2 Problematika logických hier

Logické hry sú skupinou hier, pri ktorých musí riešiteľ použiť istý špecifický postup riešenia. My sa budeme zaoberať číselným variantom týchto hier, pri ktorom sa buďto vyplňujú čísla do políčok alebo len napomáhajú k výsledku.

2.1 Nástup logických hier

Číselné logické hry vznikali už v 19. storočí v Japonsku a ďalších ázijských krajinách. Začiatkom 20. storočia sa v novinách začali sporadicky objavovať aj vo Francúzsku a Veľkej Británii. V ďalších desaťročiach sa vytratili z povedomia širokej verejnosti [2]. Až koniec 20. storočia a začiatok nového milénia zaznamenali veľký nárast popularity týchto hier. Ich výskyt v dennej tlači nasledoval záujem ľudí o logické hry na internetových stránkach. Najpopulárnejšími hrami sa stali Sudoku, Kakuro a Maľované krížovky. K obľúbenosti prispeli aplikácie, ktoré vedeli rýchle vytvárať mnoho zadaní jednotlivých hier a zároveň overovať správnosť riešenia užívateľa.

2.2 Sudoku

Úlohou je číslicami vyplniť hracie pole o šírke i výške 9 dielov. Pritom každý riadok, každý stĺpec a každá z deviatich častí veľkosti 3x3 musí obsahovať čísla od 1 po 9. Na začiatku hry je hracie pole vopred vyplnené istým počtom číslic na náhodných pozíciách (viď obrázok 1).

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

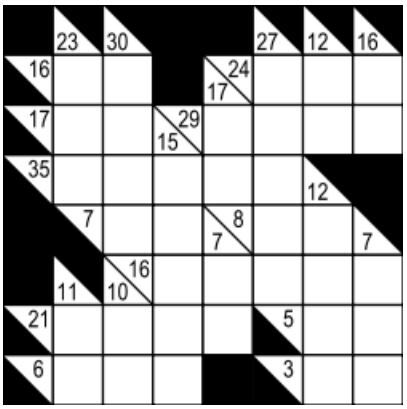
Obr. 1 - Hracie pole hry Sudoku

Vyplňovaním prázdnych políčok podľa pravidiel hry Sudoku sa nám podarí vyriešiť pole, ktoré by malo obsahovať 81 správne vyplnených čísel. Hra má mnoho variácií líšiacich sa veľkosťou hracieho poľa či kombinovaním elementov z hier Sudoku a Kakuro/Futoshiki.

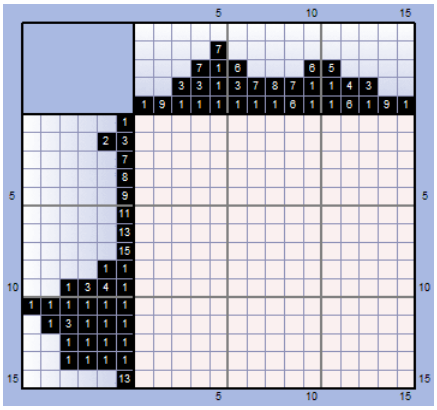
2.3 Ďalšie logické hry

Kakuro je logická hra využívajúca princípy klasických krížoviek s tým rozdielom, že namiesto slov sa do políčok vyplňujú číslice od 1 po 9. Teda nad vyplňovaným stĺpcom alebo naľavo od vyplňovaného riadku sa nachádza súčet čísel v danom stĺpci/riadku (viď obrázok 2). Je potrebné vyplniť prázdne miesta číslami tak, aby splňovali tieto súčty. Veľkosť tejto číselnej krížovky môže byť rôzna.

Nonogram (Griddlers) tiež známe ako maľované krížovky je obrázková logická hra, v ktorej sa políčka vyfarbujú alebo nechávajú prázdne pre vytvorenie výsledného obrázku. Vyfarbovanie závisí od čísiel umiestnených nad stĺpcami a vedľa riadkov (viď obrázok 3). Tieto hodnoty hovoria o počte vyplnených políčok za sebou. Napríklad čísla v riadku ‘3 4 5‘ znamenajú, že vyplnené budú tri pozície za sebou, potom štyri pozície a nakoniec päť políčok, pričom medzi týmito sériami musí byť aspoň jedna medzera. Farbiť sa môže jednou farbou alebo viacerými a toto je naznačené farbou čísiel. Tvar poľa je často štvorec, ale všeobecne môže byť rôzny.

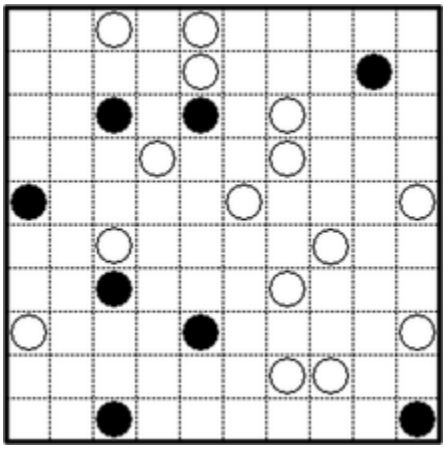


Obr. 2 - Hracie pole Kakuro



Obr. 3 - Hracie pole Griddlers

Masyu je logickým hlavolamom. Vzniklo ako hra, ktorá nepoužíva čísla či písmená. Napriek tomu si zachovala svoju hĺbku a estetický vzhľad typický pre logické hry. Hrá sa v obdĺžnikovom poli tvorenom štvorčekmi. Niektoré z týchto štvorcov obsahujú krúžok a ten je buď plný (vyfarbený) alebo prázdny. Cieľom je nakreslenie jedinej neprerušovanej čiary, ktorá správne prechádza kruhmi a nakoniec vytvorí slučku. Táto slučka musí vchádzať do každej bunky, ktorou prechádza stredom jednej zo strán tejto bunky. Vychádzať musí vždy z inej strany. Obe varianty kruhov majú rozdielne pravidlá hovoriace o tom ako slučka musí nimi prejsť [3].

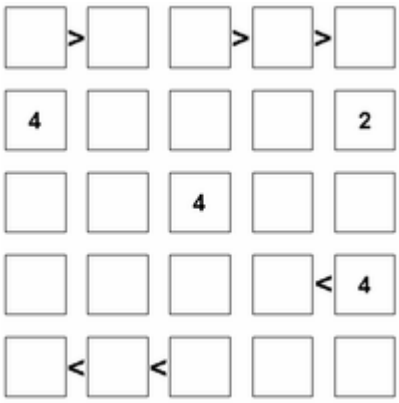

Obr. 4 - Hracie pole Masyu

3 Hra Futoshiki

V tejto kapitole sa budeme zaoberať vývojom hry futoshiki a jej pravidlami. Vysvetlíme si možné variácie a ako ovplyvňujú náročnosť riešenia. Na záver kapitoly je uvedené rozdelenie tvorby algoritmu a aplikácie.

3.1 História

Futoshiki (Nerovnosť) je logická hra z Japonska, ktorá vznikla zo známejšieho Sudoku. Zatiaľ čo si ponechala štvorcový tvar hracieho poľa, pravidlá boli doplnené o znaky nerovnosti (viď obrázok 5). Zatiaľ čo v Ázii bola známou už skôr západné štáty začali hru spoznávať až po nástupe Sudoku. Hra sa stala veľmi obľúbenou prevažne v posledných rokoch vďaka dennej tlači, v ktorej sa začala objavovať. Prvé denníky uverejňujúce túto logickú hru na svojich stranách boli vo Veľkej Británii, konkrétne The Guardian a The Daily Telegraph [1].



Obr. 5 - Hracie pole 5x5

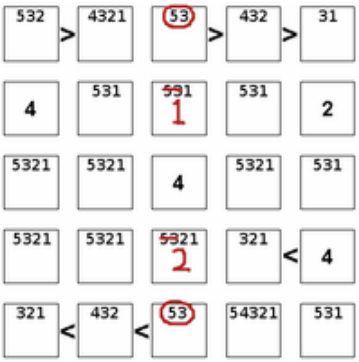
3.2 Pravidlá

Hrá sa na hracej ploche skladajúcej sa zo štvorčekov - prázdnych či obsahujúcich čísla. Medzi štvorčekmi sa nachádzajú medzery alebo znaky nerovnosti. Počet štvorčekov ovplyvňuje veľkosť plochy (viď obrázok 5). Tou je vždy štvorec o veľkosti 4x4, 5x5, 6x6 alebo viac štvorčekov (pozn. polia 3x3 a menšie sa nepoužívajú). Každý riadok a stĺpec musí obsahovať každú z číslic 1 až N, kde N je veľkosť poľa. To znamená, že v poli 5x5 sa čísla od 1 po 5 nachádzajú v každom riadku i stĺpci len raz. Tieto čísla však musia splňovať aj obmedzenia dané znakmi nerovnosti.

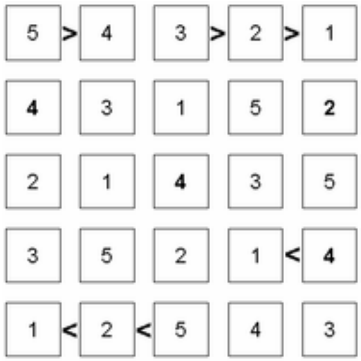
3.3 Postup riešenia

Riešenie pozostáva z určenia možných číslic pre každú pozíciu (štvorček). Každá pozícia predstavuje množinu čísel od 1 po N. Sledovaním obmedzení daných znakmi nerovnosti a vopred vyplnenými číslami môžeme tieto množiny zmenšiť. Postupným vyradzovaním číslic z každej pozície osobitne docielime v ideálnom prípade to, že nám vznikne množina s jedným prvkom. To znamená, že je možné túto pozíciu vyplniť týmto prvkom. Potom postup opakujeme so zreteľom na práve vloženú číslicu.

Ďalším dobrým postupom ako znižovať počet čísel v jednotlivých množinách je sledovanie množín kde sú len 2 možnosti [4]. Ak máme v stĺpci/riadku dve takéto pozície, môžeme číslice z nich vylúčiť vo zvyšných štvorčekoch stĺpca/riadku (viď obrázok 6). Znaky nerovnosti taktiež poskytujú oporné body pre riešenie logickej hry. Hlavne ak sú týmito znakmi zreťazené viaceré štvorce.



Obr. 6 - Vylučovanie číslic z množín



Obr. 7 - Správne riešenie hry

Opakovaním spomenutých krokov a dodržovaním pravidiel pre vyplňovanie sa dopracujeme k riešeniu, ktoré má všetky pozície vyplnené (viď obrázok 7).

3.4 Dekompozícia tvorby aplikácie

Výsledný program implementujúci hru Futoshiki musí obsahovať tieto základné časti :

- generátor pre zadania hry
- kontrolu užívateľovho riešenia
- časť uskutočňujúcu algoritmus riešiaci náhodné zadanie
- užívateľské prostredie komunikujúce s hlavným programom (triedou) - ten spája všetky časti dohromady a tým vzniká plnohodnotná aplikácia

Rozdelením tvorby na osobitné časti sa dovŕši modularita programu. To znamená, že správnym určením komunikácie hlavného (obslužného) programu je možné spracovať každý podproblém nezávisle. Umožňuje to aj možné zmeny či vylepšenia v budúcnosti bez toho, aby sa musel zložito prepisovať celý kód aplikácie.

4 Algoritmus tvorby zadania hry

Súčasťou tvorby zadania bude overovanie správnosti samotného zadania. Množstvo vopred vyplnených políčok bude závisieť od nastavenej náročnosti. Algoritmus by mal byť schopný rýchle vytvoriť zadanie, ktoré môže mať i viacero riešení.

4.1 Návrh

Algoritmus bude pracovať s dvojrozmerným poľom typu 'bunka'. Teda skladať sa bude z NxN buniek kde N je veľkosť hracieho poľa. Po vytvorení buniek sa začnú do jednotlivých políčok vkladať hodnoty.

Aby sa pole vždy vytváralo náhodne použijem triedu *Random* a budem stále generovať číslo od 1 po N. Vygenerované číslo sa vloží na pozíciu v poli a následne porovná s hodnotami v rovnakom riadku a stĺpci. Pri zhode sa vykoná kontrola či je možné vložiť do bunky aspoň jednu z číslic 1 až N a ak áno tak je vygenerované ďalšie číslo a znovu prebehne kontrola. V prípade, že nie je možné do políčka vložiť číslo tak sa resetujú čísla vyplnené v danom riadku a vyplňuje sa znova od 1. stĺpca daného riadku.

Vzniklo pole vyplnené číslami a teraz bude potrebné do neho doplniť znaky nerovnosti. Tieto znaky sa nachádzajú medzi dvoma susednými bunkami a predstavujú porovnanie ich hodnôt. Do jednorozmerného poľa sa postupne zadajú čísla reprezentujúce jeden zo štyroch možných symbolov. Zároveň sa bunkám, ku ktorým sa symbol vzťahuje nastaví premenná označujúca či v danom smere je hodnota bunky menšia alebo väčšia ako hodnota susediacej bunky. Každá bunka má štyri tieto premenné, pre každý smer jednu.

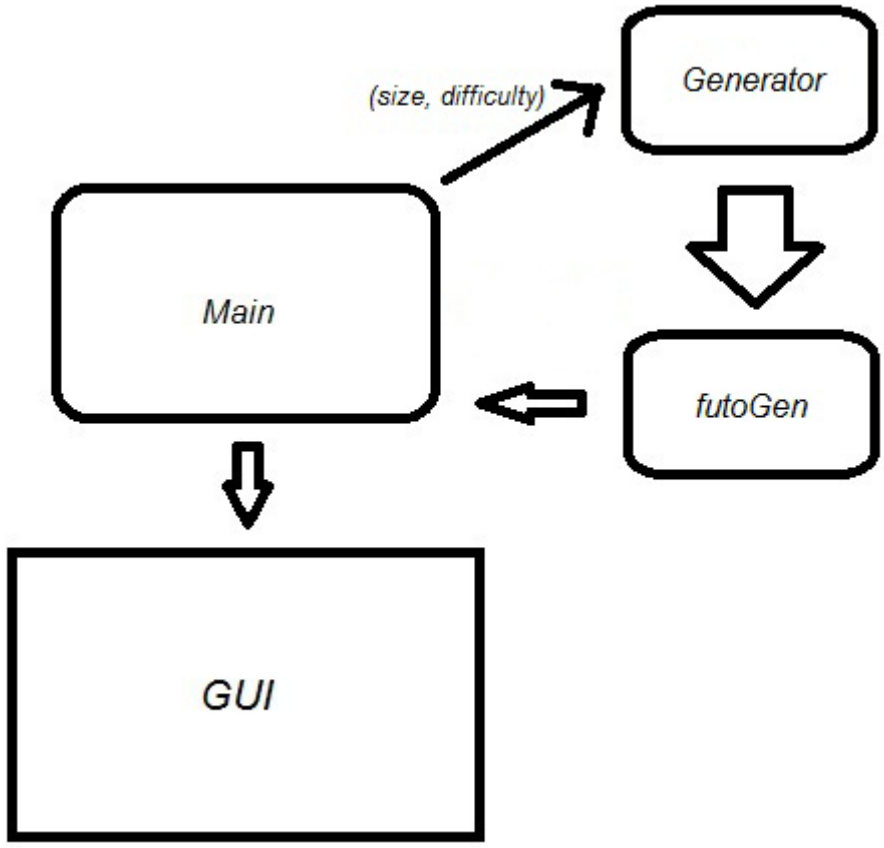
Nakoniec sa úplne vyplnené pole predá hlavnej triede. V tej sa opäť pomocou triedy *Random* náhodne vyberú z poľa prvky (čísla, znaky), ktoré sa zobrazia. Výber bude ovplyvnený len zvolenou náročnosťou. Tá rozhodne o pravdepodobnosti, že daný prvok bude zobrazený. Tak vzniká zadanie hry Futoshiki, ktoré ma určite aspoň jedno riešenie (no môže ich byť aj viacero).

4.2 Implementácia

Návrh som implementoval v programovacom jazyku Java. Pri písaní algoritmu na tvorbu zadania hry Futoshiki som vytvoril 3 triedy – *Main*, *Generator*, *Cell*. Hlavná trieda (*Main.java*) inicializuje triedu *Generator* a predá jej veľkosť (N) poľa i nastavenú náročnosť (viď obrázok 8). V triede, ktorá zabezpečuje generovanie hracieho poľa (*Generator.java*) sa inicializuje NxN buniek. Každá z týchto buniek je typu *Cell* a obsahuje jednorozmerné pole (*numArray*) o veľkosti N.

Nasleduje volanie metódy na vyplňovanie buniek (*createArray*). Bunky sa vyplňujú postupne po riadkoch. Pri každej pozícii sa volá metóda (*fillCell*), ktorá sa zaoberá vygenerovaním hodnoty i jej vložením do bunky určenej súradnicami. Číslo sa vygeneruje v rozsahu od 1 po N. Potom sa vloží ako hodnota danej bunke (*value*) pričom sa odstráni z možných hodnôt ostatným bunkám v riadku a stĺpci. Na to slúži spomenuté pole v triede *Cell*, ktoré obsahuje všetky možné hodnoty aké bunka môže nadobudnúť.

V prípade, že je niektorá hodnota (index v poli *numArray*) označená ako nevhodná (nastavením hodnoty v danom indexe na 0 alebo N+1) nie je možné ju použiť ako hodnotu bunky. V prípade, že vygenerované číslo nie je možné vložiť prebehne kontrola či momentálne vyplňovaná bunka je schopná nadobudnúť niektorú z platných hodnôt. Táto kontrola prebehne pomocou metódy *checkCell*, ktorá je metódou triedy *Cell* a kontroluje pole *numArray*.



Obr. 8 - Inicializácia generátoru a predanie výsledku hlavnému programu

Ak kontrola objaví, že políčku nie je možné prideliť žiadnu hodnotu vymažú sa všetky vložené čísla v danom riadku. Súčasne je každé zmazané číslo opäť dostupné v stĺpci, z ktorého bolo odobraté. Po resetovaní sa opäť vyplňuje spracovávaný riadok od 1. stĺpca. Pri tvorbe tejto časti sa vyskytol problém, o ktorom som v návrhu neuvažoval. Ten spočíva v tom, že pri vyplňovaní predposledného riadku môže dôjsť k situácii keď už žiadne usporiadanie čísel v riadku poslednom neumožní vyplnenie všetkých buniek. Preto existuje špeciálne pravidlo, ktoré ak zistí neschopnosť vyplnenia posledného políčka (teda na pozícii [N,N]) tak zresetuje nielen riadok posledný, ale i predposledný. Všetky zmazané číslice sú opäť dostupné pre riadok/stĺpec kde boli odstránené. Pokračuje sa ďalej vo vyplňovaní od 1. stĺpca predposledného riadku [7].

Súčasťou každej bunky sú aj štyri premenné (*up*, *down*, *left*, *right*), ktoré obsahujú informácie o susedných bunkách. Konkrétne či hodnoty susedných buniek sú väčšie alebo menšie. Ak je premenná nenastavená (jej hodnota je 0) znamená to, že tým smerom nie je znak nerovnosti.

V procese generovania sa už vyplnili čísla do všetkých políčok. Teraz je potrebné doplniť znaky nerovnosti medzi niektorými políčkami. Výber bude rozhodovaný triedou *Random*, ktorá vygeneruje číslo a ak to číslo je väčšie ako určený prah (náročnosť) znak nerovnosti sa vloží. Okrem vloženia symbolu sa nastavia aj premenné susediacich políčok. Ktoré sa nastavia a ako, rozhoduje pozícia a typ symbolu.

V závere hlavná trieda volá obsah buniek i obsah poľa symbolov nerovnosti a vykresľuje ho. Symboly sa vykreslia všetky, ale čísla sa zobrazia len ak náhodne vygenerované číslo je väčšie ako prah (náročnosť). Obdobne ako to bolo pri symboloch nerovnosti. Dôvodom pre nutnosť selekcie znakov nerovnosti už v triede *Generator* je zjavná z ďalšej kapitoly, ktorá sa zaoberá kontrolou užívateľom vyplneného riešenia.

Vzniká hracie pole, ktoré ma zaručene aspoň jedno správne riešenie. Keďže správnych riešení môže byť viac nie je možné použiť vygenerované pole (pozn. vygenerované pole nie je zhodné s poľom zobrazovaným) na kontrolu čísel vyplnených užívateľom. Preto problém kontroly správnosti riešenia musí byť ošetrený samostatným algoritmom. Jeho riešenie je obsahom ďalšej kapitoly.

4.3 Výhody a nevýhody riešenia

Pri efektívnosti algoritmu je potrebné vziať do úvahy časovú a pamäťovú náročnosť. Pri algoritme bolo nutné uvážiť každý možný postup riešenia a jeho dopad na efektivitu. V istom bode tvorby vznikla situácia kde som mohol vybrať či bude program časovo alebo pamäťovo menej náročný. Keďže generovanie poľa bolo už dostatočne rýchle zvolil som druhú možnosť. Tá sa prejavuje použitím triedy *Random* pri generovaní čísel vkladaných do buniek. Tento problém bolo možné riešiť spôsobom pamätania si použitých stavov pomocou premenných a metód. Tento spôsob je pamäťovo náročnejší a tak je nevhodný v prípade, že hracie pole sa generuje už dostatočne rýchle.

Ďalšou spornou časťou je návrat pri stave kde nie je možné už bunku vyplniť. Tá však bola riešená s ohľadom na obe zložky (časovú i pamäťovú). Je možné sa vrátiť a hneď generovať nové pole. Nie je to však nutné kvôli resetovaniu len súčasného riadku a pridaniu spomenutej výnimky (vrátenie sa na začiatok predposledného riadku). To zabezpečí, že generovanie sa nikdy nezasekne. No zároveň sa nezhoršuje ani časová náročnosť, ktorá by sa pri tvorbe úplne nového poľa zvýšila. V tejto časti je i tretia možnosť – pamätať si použité stavy buniek. To však zvýši nielen pamäťovú, ale i časovú náročnosť čo by malo citeľný dopad na efektivitu generovania poľa.

Mnou zvolené postupy v jednotlivých častiach algoritmu generovania zadania sú efektívne. Hoci nie všetky dosahujú najlepšiu časovú či pamäťovú zložitosť, kombináciou týchto zložitostí dosiahneme efektívne pracovanie algoritmu v dobrom čase bez použitia veľmi veľkého množstva pamäte či iných prostriedkov.

5 Algoritmus pre kontrolu riešenia hry

Po vyplnení hracieho poľa užívateľom je nutné previesť kontrolu tohto riešenia. Táto kontrola by mala byť časovo nenáročná a uskutočniť by sa mala v dvoch krokoch. Prvým je kontrola čísel v riadku a stĺpci. Druhým kontrola buniek medzi ktorými sa nachádza znak nerovnosti.

5.1 Návrh

Použije sa dvojrozmerné pole už inicializované generátorom hracieho poľa. Využijú sa všetky nastavenia v bunkách, vrátane označení indikujúcich znak nerovnosti v okolí bunky. Jedine obsah políčok, ktoré sa dajú prepisovať (na začiatku hry sú prázdne) sa predá na odpovedajúcu pozíciu v poli. Pritom sa prepíše pôvodná hodnota políčka a následne sa táto nová hodnota odoberie z poľa možných hodnôt buniek v rovnakom riadku/stĺpci. Ak dôjde k situácii, že daná hodnota už zakázaná je, algoritmus okamžite označí riešenie užívateľa za nesprávne. Len úplný prechod poľom označí toto riešenie za správne.

5.2 Implementácia

Nie sú vytvorené žiadne nové triedy, algoritmus pracuje len s troma vytvorenými. Vygenerované bunky (ich hodnoty a hodnoty ich premenných) sa predajú metóde v hlavnej triede (*getResult*). Táto metóda načíta čísla, ktoré boli doplnené do poľa užívateľom. Pri chybe vracia číslo 1 a tým sa označí riešenie za nesprávne. Ak táto časť kontroly prejde tak sa vykoná ďalšia časť – odstránenie novej vloženej (užívateľskej) hodnoty z polí možných hodnôt ostatným bunkám v riadku/stĺpci. Pokiaľ k odstráneniu nemôže dôjsť znamená to, že dané číslo sa už v riadku/stĺpci nachádza. V tomto prípade sa vracia chybový kód a riešenie sa stane neplatným.

Ak prvé dva kroky nevrátia chybový kód vykoná sa posledná kontrola. Tá kontroluje bunky, konkrétne ich špeciálne premenné označujúce znak nerovnosti v danom smere. Tieto premenné môžu mať 3 stavy – žiadny znak, znak (bunka je menšia), znak (bunka je väčšia). Práve kvôli tomu, že tieto hodnoty sa použijú pri kontrole riešenia nebolo možné vložiť znaky nerovnosti medzi všetky bunky a následne len niektoré znaky nezobraziť (tak ako je to pri číslach). Tretia kontrola tak spočíva v tom, že ak nájde premennú nastavenú na stav ‘znak’ skontroluje hodnotu tejto premennej. Následne porovná hodnotu súčasnej bunky s hodnotou bunky v smere premennej. Ak sa porovnanie nezhoduje so stavom premennej vracia sa chybový kód a riešenie je nesprávne.

V prípade, že riešenie vložené užívateľom úspešne prejde všetkými kontrolami vracia funkcia číslo 0. To indikuje, že riešenie je správne vyplnené.

5.3 Výhody a nevýhody riešenia

Jednou nevýhodou tohto riešenia je strata istej časti modularity. Algoritmus kontroly sa vykonáva v rovnakej triede ako algoritmus generovania hracieho poľa takže úpravy sú zložitejšie. Hlavne zmena generátoru ovplyvní výrazne tento algoritmus.

Naopak efektivita programu narastá pretože nie je nutné inicializovať nové pole a nenastavujú sa všetky hodnoty buniek. Taktiež vďaka špeciálnym premenným a metódam buniek sú kontroly jednoduché. Práve toto umožňuje úplnú kontrolu užívateľovho riešenia len v troch rýchlych krokoch. Celá kontrola sa tak stáva len vyhodnotením porovnaní jednotlivých premenných.

6 Algoritmus pre riešenie hry

Riešenie zadania hry Futoshiki sa musí vykonávať v krokoch. Tým sa od seba oddelia jednotlivé pravidlá, ktoré treba dodržiavať. Prvým krokom bude kontrola vyplnených čísel v riadku/stĺpci pričom sa tieto čísla vylúčia z množín jednotlivých pozícií. Druhý krok by mal kontrolovať znaky nerovnosti a taktiež vylúčiť nevhodné čísla z množín. Tretím krokom bude 'hádanie' čísla na istej pozícii a následný návrat na prvý krok.

6.1 Návrh

Prvým krokom bude (podobne ako v generátore) vytvorenie poľa o veľkosti NxN. Ďalej nasleduje načítanie políčok, ktoré sú vyplnené (buď užívateľom alebo generátorom) a vloženie týchto hodnôt do poľa v riešiteľovi. Pri vložení hodnoty do bunky sa táto hodnota odoberie z možných hodnôt ostatným bunkám v rovnakom riadku či stĺpci. To zjednoduší prácu algoritmu, ktorý môže pracovať s už korektne nastavenými bunkami.

Načítanie znakov nerovnosti bude druhým krokom. Bunky sú schopné uchovávať informácie o znakoch nerovnosti v ich okolí (pozn. bližšie informácie sú v predchádzajúcej kapitole). Po načítaní znaku sa tak len určí jeho pozícia vzhľadom na pole a okolité bunky sa náležite nastavia. Súčasne ak je jedna z okolitých buniek vyplnená (obsahuje číslo) prázdne bunky si nastavia svoje dosiahnuteľné hodnoty vzhľadom na hodnotu tejto bunky a znak nerovnosti. Toto opäť urýchľuje samotný algoritmus riešenia hry pretože zúži možný výber čísel pre niektoré bunky.

Pretože užívateľ môže vytvoriť už vopred neriešiteľné pole prebiehajú kontroly pri vkladaní hodnôt a nastavovaní buniek podľa znakov. Každá kontrola sa uskutočňuje keď sa mení nastavenie bunky (jej hodnota či iné premenné). V prípade, že nastavenie nie je možné vykonať vráti sa chybový kód a algoritmus pre riešenie hracieho poľa sa nikdy nespustí. Jedine ak dôjde k spusteniu algoritmu sa toto zadanie môže považovať za správne.

Samotný algoritmus bude pracovať na princípe 'najlepšej voľby'. Každá bunka má v momente spustenia algoritmu určitý počet možných hodnôt. Zavolá sa metóda, ktorá prejde celým poľom a zistí pozíciu bunky s najmenej možnými hodnotami. Ideálnym prípadom je jedna možná hodnota kde sa bunke táto hodnota pridá. V prípade, že sa musí vybrať jedna z viacerých hodnôt bunky označia sa všetky ďalšie vyplňované hodnoty buniek za 'uhádnuté'.

V momente keď algoritmus nemôže doplniť hodnotu do bunky sa v poli skontrolujú všetky bunky. Ak niektorá nemá pridelenú hodnotu všetky hodnoty buniek nastavené ako 'uhádnuté' sa vymažú. Potom sa znova vyplňuje pole. Tento postup sa opakuje kým nie je celé pole vyplnené. Vyplňovanie sa uskutočňuje pomocou triedy *Random*, ktorá vygeneruje náhodné číslo z možných hodnôt bunky. Keď je pole kompletne vyplnené hodnoty si prevezme hlavná trieda.

6.2 Implementácia

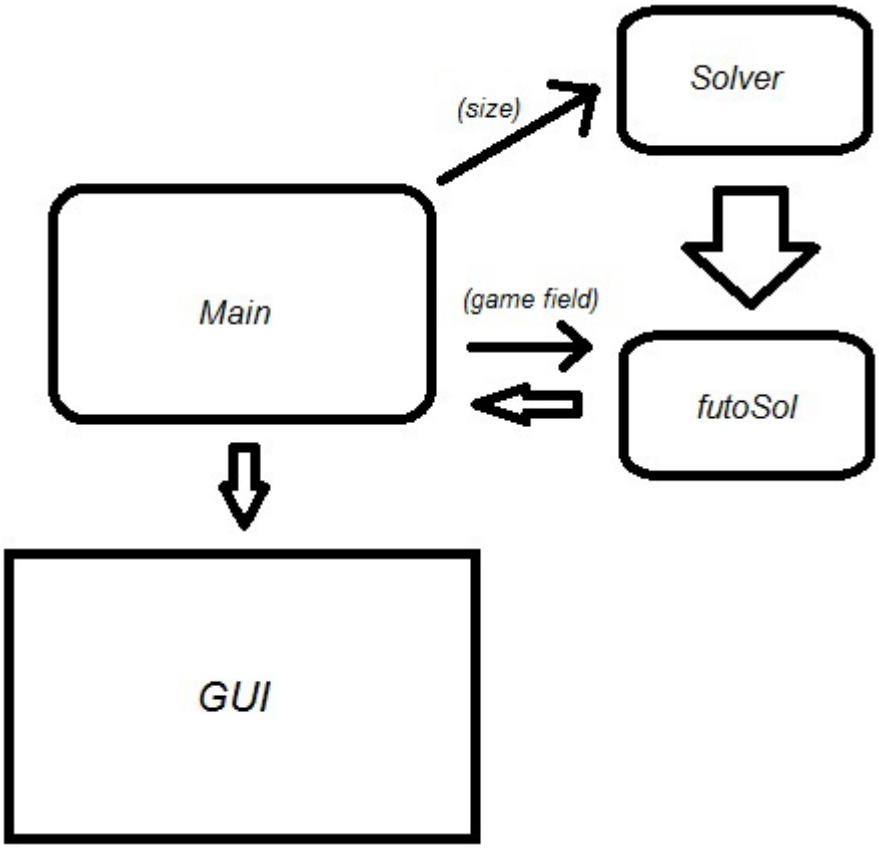
Nutným úkonom je vytvorenie vlastného dvojrozmerného poľa veľkosti NxN a typu *Cell*. Kvôli možnosti, že hracie pole bolo vytvorené užívateľom nie je možné použiť pole už inicializované v generátore. Preto vzniká nová trieda (*Solver.java*). Táto trieda vykonáva nielen riešiaci algoritmus, ale aj prvotné nastavenia v bunkách pri načítavaní hracieho poľa. Má tri verejné metódy (*getFill*, *getHighLow* a *initResult*) volané hlavnou triedou. Prvé dve rozhodujú o vykreslení riešenia alebo rozhodnú o neplatnosti zadaného poľa (pozn. táto možnosť môže nastať len pri užívateľovom hracom poli). Tretia je výkonnou časťou algoritmu.

V hlavnej triede sa zavolá metóda triedy *Solver* vyplňujúca čísla z hracieho poľa (viď obrázok 9). To môže byť vytvorené užívateľom alebo aj generátorom popísaným v kapitole 4. Pri vyplňovaní sa bunky, ktorým sa priradí hodnota označia za nemenné (*fixed*). Zároveň sa podobne ako pri predošlých algoritmoch nastavia polia možných hodnôt buniek v rovnakom riadku/stĺpci. Toto nastavovanie umožní rýchlejší prechod algoritmu. Taktiež tu môže prebehnúť prvá kontrola, lebo ak nie je možné vykonať jedno s nastavení premenných vráti metóda chybový kód. Následné správanie programu je už popísané vyššie, v návrhu algoritmu.

Ďalšia metóda načítava znaky nerovnosti v poli. Pomocou údajov o ich pozíciách nastaví okolitým bunkám špeciálne premenné spomínané už v generátore hracieho poľa. Tieto premenné informujú o porovnaní hodnôt susedných buniek medzi ktorými je znak nerovnosti. Podľa typu symbolu sa vykonávajú porovnania čísel v políčkach či dodržujú pravidlá hry Futoshiki. Okrem toho sa opäť nastavujú polia možných hodnôt, tento krát len pre susedné bunky. Ak pri tomto procese nie je možné vykonať nastavenie znovu sa vracia chybový kód. V opačnom prípade sú bunky v poli nastavené pre riešiaci algoritmus [7].

Algoritmus riešenia hracieho poľa sa vykonáva v cykle. Tento cyklus zastaví len správne vyriešenie hracieho poľa, čo bol dôvod kontroly a správneho nastavenia buniek vopred. Na začiatku každého cyklu sa kontroluje pole. V prípade, že všetky bunky majú pridelenú hodnotu cyklus sa ukončí. Ďalej sa nastavujú bunky, ktoré majú v okolí znak nerovnosti. V prvom prechode je tento krok zbytočný no v ďalších berie do úvahy algoritmom vyplnené čísla.

Prebieha kontrola buniek a zisťovanie ich možných hodnôt. Bunka s najmenším počtom je vyplnená číslom. Ak je viac buniek s najmenším počtom vyplní sa tá prvá (pozn. pole sa prechádza po riadkoch). Výnimkou je ak majú tieto bunky len jednu možnosť, v tom prípade sa vyplnia všetky naraz a sú označené ako nemenné. Dosadzovanie čísel sa deje za pomoci triedy *Random,* podobne ako pri generátore opísanom vo štvrtej kapitole. A opäť sa dosadená hodnota odstráni z možných hodnôt ostatným bunkám v riadku/stĺpci.



Obr. 9 - Náčrt implementácie riešiteľa do hlavného programu

V prípade, že najmenší počet možných hodnôt je 2 alebo viac nastáva v algoritme špeciálny stav (*tagged*). Tento stav spôsobuje, že všetky bunky od nastavenia tohto stavu sú menné. Okrem toho sa mení aj spôsob akým je hodnota vyplnenej bunky nastavená ako nevhodná ostatným bunkám v riadku/stĺpci. Tieto nastavenia sú dôležité pre prípad zresetovania poľa. Keď algoritmus nemôže vložiť číslo do žiadnej bunky a súčasne je aspoň jedna bunka nevyplnená, nastaví sa do stavu resetovania. V tomto stave sa zmažú hodnoty všetkých buniek, ktoré sú označené ako nemenné. Súčasne sú v celom poli zmazané všetky špeciálne označené hodnoty v poliach možných hodnôt (*numArray*). Tým sa pole navráti do stavu kedy nastalo prvý krát hádanie hodnoty bunky. Po viacerých pokusoch o vyriešenie hracieho poľa algoritmus nastaví správne hodnoty jednotlivým políčkam. Hlavná trieda už môže zobraziť vytvorené riešenie.

6.3 Výhody a nevýhody riešenia

Úvodné nastavovanie buniek dosahuje čo možno najlepšiu efektivitu. Nevolajú a nenastavujú sa premenné, ktoré nie sú nutné pre práve vykonávanú časť. V prípade nutnosti zvýšenia počítadiel sa preskočia všetky kontroly a žiadna z premenných bunky sa nenastavuje.

Problém výberu a vkladania hodnôt je riešený obdobne ako pri generátore. Riešený je teda za pomoci kompromisu medzi postupmi s najlepšou časovou a najlepšou pamäťovou zložitosťou. Nastavovaním buniek ako nemenné sa znižuje čas, ktorý algoritmus strávi pri riešení poľa po resetovaní. Nemusí sa zaoberať bunkami, ktorých hodnoty sú jednoznačné. Pamäťová náročnosť sa znižuje vynechaním štruktúry pamätajúcej si všetky hodnoty poľa vyplnené pred jeho resetovaním. Namiesto toho sa podobne ako pri generátore používa trieda *Random*, ktorá náhodne vyberie číslo a vloží ho ak je to povolené.

Algoritmus je tak maximálne efektívny i napriek tomu, že jeho jednotlivé zložky nie sú. Čas riešenia je vyhovujúci pretože pri najväčšej veľkosti poľa (9x9) netrvá dlhšie ako 3 sekundy a priemerná doba riešenia je len 1 sekunda. Preto bolo dôležité sa opäť sústrediť na množstvo zaberaných zdrojov. Pomocnými premennými a stavmi buniek sa umožnilo prenechanie len rozhodovacích úkonov algoritmu. Tieto premenné a stavy zaberajú omnoho menej zdrojov než by to bolo u štruktúry uchovávajúcej vykonané ťahy a vloženia hodnôt.

Efektivita riešenia sa zvyšuje pri zvýšení veľkosti poľa. To je dôsledkom použitia triedy *Random*. Pri menších poliach je síce aj menej možných stavov bunky, ale hrozí opakovanie vyplňovania. Zatiaľ čo pri väčších poliach toto opakovanie je menšie kvôli väčšiemu výberu hodnôt. To však neznamená rýchlejšie vyriešenie väčšieho poľa pretože počet resetovaní kvôli nesprávnemu uhádnutým hodnotám sa zvyšuje s počtom možných hodnôt aké bunka môže nadobúdať. Okrem toho je samozrejme nutné vyplniť viac buniek.

7 Užívateľské prostredie hry

Tvorba užívateľských prostredí vyžaduje neustály cyklus, v ktorom sa opakuje vytváranie prostredia aplikácie a testovanie tohto prostredia užívateľmi. Cieľom je dosiahnutie istej úspešnosti daného riešenia. Toto testovanie môže byť spojené s testovaním funkčnosti programu, pričom sa overuje nielen funkčnosť, ale aj logickosť a pohodlnosť výsledného dizajnu aplikácie.

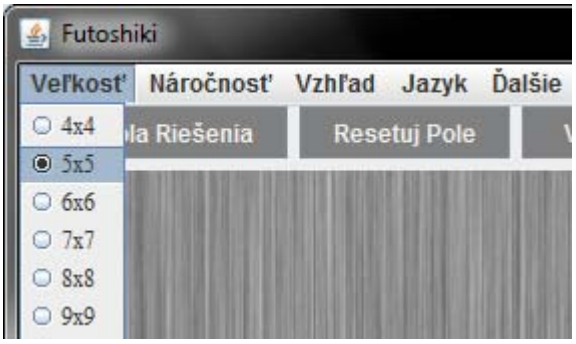
7.1 Návrh

Aplikácia bude mať hlavné okno, ktoré bude obsahovať riadiace prvky pre generovanie hracieho poľa a jeho riešenie. Taktiež tu budú rôzne ponuky na zmenu nastavení [6]. Hlavnou časťou však bude plocha, na ktorej sa bude generovať hracie pole o zvolenej veľkosti. Toto pole sa bude skladať z malých štvorčekov (buniek). Na každú bunku bude možné kliknúť a doplniť do nej číslo. V prípade tvorby vlastnej hry bude možné kliknúť aj do priestoru medzi bunkami a vložiť do neho symbol nerovnosti. Riadiacimi prvkami budú tlačidlá, ktoré po stlačení vykonajú určitú úlohu (vygenerovanie, kontrolu, vyriešenie či vymazanie poľa) a zobrazia výsledok tejto úlohy buď v hracom poli alebo v novom okne. Okrem veľkosti poľa bude možné nastaviť aj jeho náročnosť. Čísla doplnené užívateľom či samotnou aplikáciou sa budú farebne líšiť od ostatných (aj od seba navzájom).

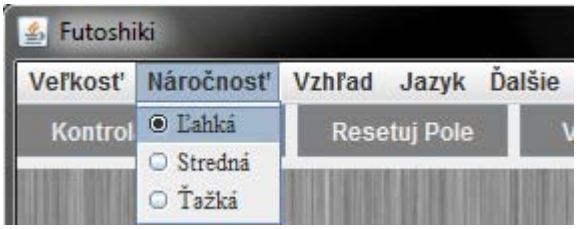
7.2 Proces tvorby

Prvým krokom bolo vytvorenie pomenovaného okna, ktoré obsahuje hracie pole. Veľkosť poľa bola zatiaľ nemenná a nastavená na 5x5. Náročnosť bola tiež nastavená len na tú najľahšiu. V tejto fáze prebiehala synchronizácia s generátorom poľa. Hlavnou úlohou bolo nastavenie obsahu buniek podľa hodnôt generátora a vykresľovanie tohto obsahu do okna [8].

Nasledovalo doplnenie ponúk pre nastavenie náročnosti a veľkosti hracieho poľa. Sú tri stupne náročnosti – ľahká, stredná a ťažká (viď obrázok 11). Škála možných veľkostí poľa je od 4x4 až po 9x9 (viď obrázok 10). Základom (t.j. vopred nastavené) však ostáva pole 5x5 na ľahkej úrovni. Pri zmene nastavenia sa ihneď generuje nové hracie pole. Hlavné okno bolo doplnené o tlačidlo kontrolujúce správnosť poľa. Po jeho stlačení sa vykoná kontrola hracieho poľa a zobrazí sa nové okno. Ak je pole správnym riešením hry Futoshiki zobrazí sa správa o úspešnosti riešenia. V opačnom prípade obsahuje nové okno zápornú správu.



Obr. 10 - Ponuka nastavení veľkosti poľa



Obr. 11 - Nastavenie náročnosti poľa

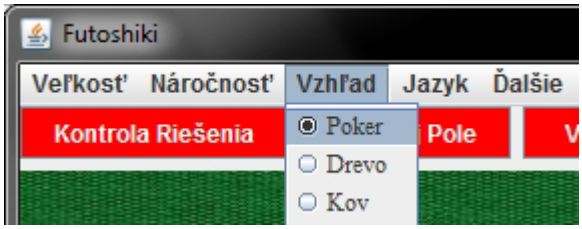
Aplikácia je ďalej doplnená o dve tlačidlá. Prvé rieši zadané hracie pole a druhé vytvorí prázdne hracie pole, vhodné pre zadanie užívateľovho hracieho poľa. Po stlačení prvého tlačidla sa inicializuje riešiteľ a výsledok sa zobrazí ako čísla doplnené do hracieho poľa. Tieto čísla sú farebne odlíšené od zvyšku poľa. Tvorba vlastného poľa umožní doplňovanie čísel i znakov do buniek a priestorov medzi nimi. Táto verzia je už predložená skúšobnej vzorke užívateľov.

Užívatelia sa vyjadrili, že im chýbajú dve prvky. Tými sú resetovanie poľa (uvedenie poľa do základného stavu po vygenerovaní) a generovanie nového poľa. Implementácia týchto funkcií do programu bola vykonaná a obom bolo pridelené vlastné tlačidlo. Resetovanie je vymazanie obsahu buniek vyplnených užívateľom alebo riešiteľom aplikácie. Sú farebne odlíšené od zvyšku hracieho poľa a tak je jednoduché ich určiť a vyprázdniť. Generovanie hracieho poľa nie je novou funkciou (vykonáva sa pri otvorení aplikácie a pri zmene nastavení). Preto je tlačidlu pridelená už existujúca funkcia. Doplnil som novú ponuku, ktorá mení súčasný vzhľad aplikácie. Pri výbere položky tejto ponuky sa farby jednotlivých elementov zmenia. Obsah okna či hracieho poľa sa však nezmení. Opäť je aplikácia otestovaná vzorkou ľudí.

Ďalšie testovanie odhalilo nepohodlnosť súčasného vkladania čísel. Okrem toho bolo požadované zobrazenie pravidiel hry Futoshiki a pomocník ovládania aplikácie. Nová ponuka teda obsahuje dve položky – odkaz na pomocníka (manuál) a odkaz zobrazujúci okno s pravidlami hry. Formát a obsah pomocníka je možné si pozrieť v prílohách. Vkladanie obsahu (čísel) do buniek už nevyžaduje vstup z klávesnice [9]. Pri kliknutí na zvolenú bunku sa táto bunka vyznačí ružovou farbou. Súčasne sa zobrazí nové okno. V tom bude na výber jedno z N čísel (N je veľkosť poľa) alebo vymazanie vloženej hodnoty. Výber je realizovaný tlačidlami a po zvolení jedného z nich sa toto okno zavrie. Zároveň je do vyznačenej bunky vložená hodnota alebo je hodnota z bunky odstránená. Podobné okno s výberom je zobrazené aj pri vkladaní znakov medzi bunky. Toto okno obsahuje štyri symboly a možnosť vymazania vloženého znaku. Tretí krát je aplikácia predstavená vzorke užívateľov a z návrhov a pripomienok je vytvorená už záverečná podoba aplikácie.

7.3 Finálna implementácia

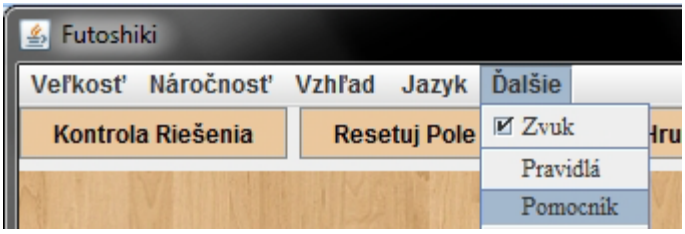
Pri spustení programu sa vygeneruje hracie pole hry Futoshiki. To sa následne aj zobrazí v hlavnom okne spolu s ovládacími prvkami aplikácie – tlačidlami a ponukami (viď obrázok 16). Prvé dve ponuky (veľkosť a náročnosť) pri zmene nastavenia vygenerujú a vykresľujú hracie pole. Zatiaľ čo generovanie má na starosti trieda *Generator* (postup je popísaný v kapitole 4), vykresľovanie obsahu panelu obsluhuje metóda *recolor*. Tá nastavuje farby jednotlivých prvkov aplikácie (buniek, panelov, tlačidiel, apod.). Tieto farby sú závislé od nastaveného vzhľadu – tretej ponuky. Na výber sú tri štýly – drevený, kovový a pokrový (viď obrázok 12). Pri každom sa menia nielen farby, ale aj pozadie panelu s hracím poľom.



Obr. 12 - Výber vzhľadu aplikácie

Ďalšou ponukou je nastavenie jazyka aplikácie (viď obrázok 13). V prípade zmeny sa zavolá metóda *changeLang*, ktorá nastaví všetok text podľa zvoleného jazyka. Poslednou dostupnou ponukou je menu obsahujúce odkaz na pomocníka a odkaz zobrazujúci okno s pravidlami hry. Okrem týchto dvoch položiek sa tu nachádza aj položka zapínajúca a vypínajúca zvuky v hre (viď obrázok 14). Zvuky sú priradené k väčšine prvkov a činností aplikácie.

Obr. 13 - Ponuka možných jazykov



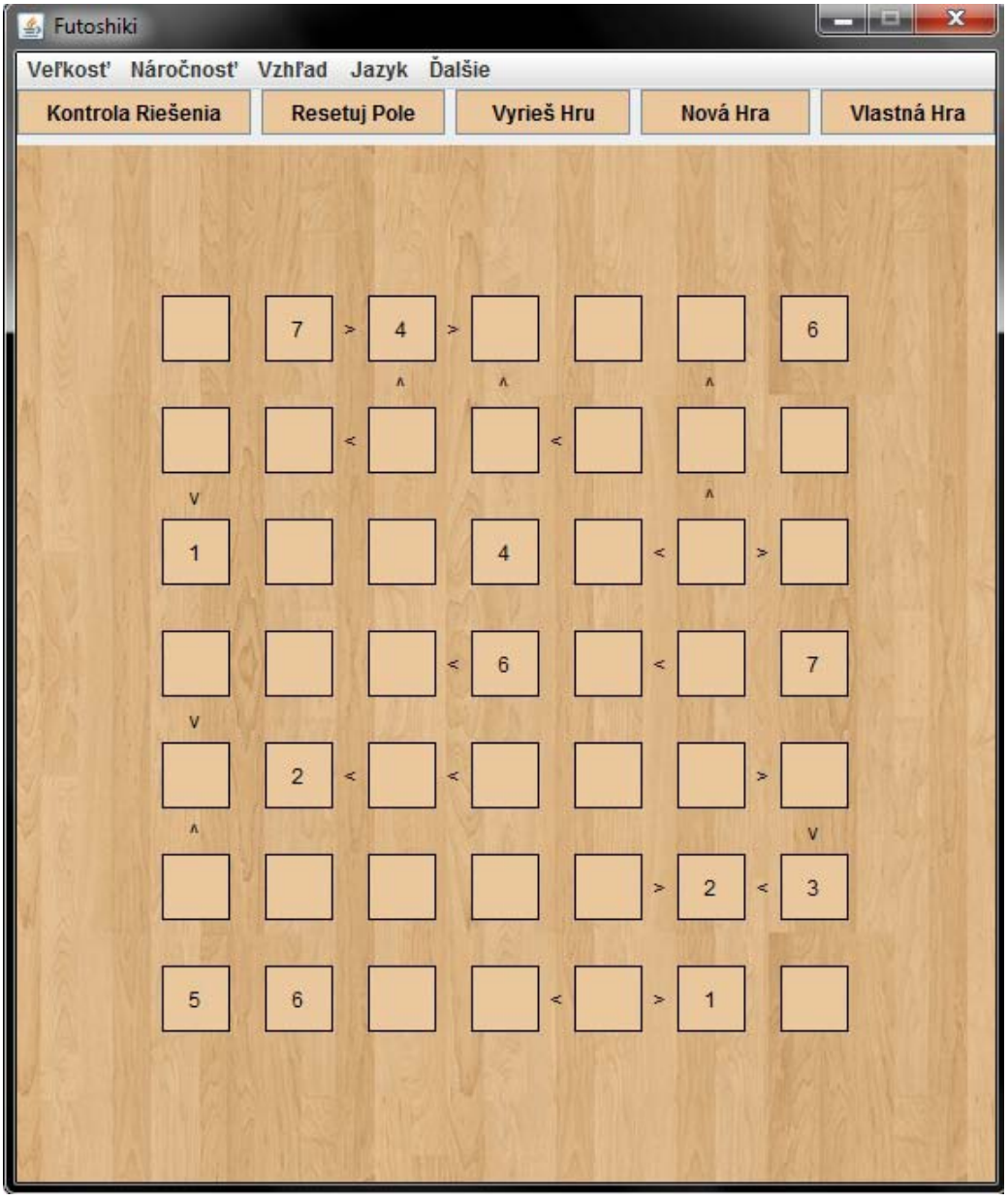
Obr. 14 - Odkazy na pravidlá a pomocníka, nastavenie zvuku

Základné herné prvky sú dostupné pomocou piatich tlačidiel vo vrchnej časti obrazovky (viď obrázok 16). Tieto tlačidlá sú :

- **Kontrola Riešenia** - prevedie sa po stlačení odpovedajúceho tlačidla v okne (metóda *getResult*). Výsledkom tohto úkonu je zobrazené okno, ktoré referuje o úspešnosti kontroly (t.j. správnosť zadaného riešenia).
- **Resetuj Pole** - tlačidlo pre nastavenie poľa do jeho pôvodného tvaru pred doplňovaním čísel (metóda *resetFilled*).
- **Vyrieš Hru** - vyrieši dané hracie pole volaním triedy *Solver* a zobrazením riešenia do tohto poľa.
- **Nová Hra** - jeho úlohou je generovanie nového hracieho poľa. Volá sa tá istá metóda ako pri zapnutí aplikácie.
- **Vlastná Hra** - Posledným tlačidlom je tvorba vlastného hracieho poľa. To zaobstaráva metóda *resetAll*. Pri tvorbe vlastného zadania sú pre vyplňovanie dostupné aj medzibunkové priestory.

Obr. 15 - Výber hodnoty do bunky

Finálnou časťou rozhrania je vkladanie samotných hodnôt (čísel a znakov) do hracieho poľa. Pri vygenerovanom poli je možné doplňovať len čísla, pri vlastnom zadaní i znaky. Po kliknutí na priestor v poli (napr. bunku) sa zobrazí okno s výberom (viď obrázok 15). Po zvolení požadovanej hodnoty sa okno zavrie a táto hodnota je vložená do priestoru, na ktorý bolo kliknuté. Ten je počas výber vyznačený ružovou farbou. Vloženú hodnotu je možné taktiež vymazať pomocou tlačidla pre mazanie obsahu. Toto tlačidlo sa nachádza v okne výberu. Pre pohodlnejšie ovládanie hry nie je možné kliknúť na čísla, ktoré už v danom riadku či stĺpci už sú.



Obr. 16 - Hlavné okno aplikácie

Bol vložený aj prvok ukladania nastavení. Pri zmene niektorého nastavenia sa zmena automaticky uloží do súboru (*settings.xml*). Pri budúcom spustení aplikácie sa načítajú nastavenia a okno programu sa zobrazí podľa nich. Na ukladanie a načítavanie nastavení boli použité pomocné knižnice pre prácu s xml dokumentmi. Knižnica *dom4j* dopomáha tvorbe xml [10]. Pomocou knižnice *jaxen* je zase možné prechádzať xml súbor po elementoch [11]. Aplikácia je predbežne hotová a nasleduje testovanie na možné chyby a preklepy. Zisťuje sa úspešnosť aplikácie u užívateľov (pozn. vzorka je väčšia ako pri predošlých testoch).

7.4 Úspešnosť aplikácie

Tvorba užívateľského rozhrania vyžaduje neustále testovanie aplikácie a získavanie poznatkov z týchto testov. Je to proces, ktorý ukončí až dosiahnutie požadovanej (dostačujúcej) úspešnosti v hodnoteniach od užívateľov. Testov musí byť viacero a po každom sa vykonajú zmeny reflektujúce snahu vyhovieť požiadavkám užívateľov [5].

Po dovŕšení verzie pokladanej za finálnu je vhodné zistiť spokojnosť, pohodlnosť či logickosť výsledného produktu. Najvhodnejšími spôsobmi sú anketa alebo dotazník. V tých sú priame hodnotenia jednotlivých častí aplikácie a tak je možné presne určiť percentuálne úspešnosti každej z nich. Ja som zvolil jednoduchý dotazník, ktorý vyplnili všetci užívatelia ktorým bola sprístupnená aplikácia Futoshiki. Formát dotazníku je možné zhliadnuť v prílohách.

Výsledok ukázal, že užívatelia sú s aplikáciou spokojný (viď tabuľku 1). Rozhranie je pre nich pekné na pohľad, ale aj logické. Finálnu verziu dokázal ovládať každý bez nutnosti vysvetľovania (pozn. aj vďaka manuálu). Najväčší úspech mal výber viacerých vzhľadov a zvuky. Žiadna časť nebola označená ako nevyhovujúca či majúca výhrady. Najslabšie hodnotenou položkou dotazníka bola forma/umiestnenie jednotlivých častí (viď tabuľku 2). Keď sa výsledky (hodnotenia jednotlivých častí) prepočítajú vychádza 91,7 percentná celková užívateľská spokojnosť s aplikáciou. Tento údaj je úspešnosťou mojej aplikácie. Tá presahuje mnou požadovanú úspešnosť 85% a tak je možné považovať tvorbu aplikácie za ukončenú.

Percentuálne hodnotenie úspešnosti častí	
Tlačidlá	91,7%
Okno hry	91,7%
Hracie pole	83,3%
Nastavenia	93,8%
Pozadie okna	83,3%
Farby	98,0%
Zvuky	100,0%

Tab. 1 - Úspešnosť jednotlivých častí aplikácie

Percentuálne hodnotenie úspešnosti položiek	
Vzhľad	91,7%
Forma/Umiestnenie	79,2%
Ovládateľnosť	95,0%
Celkový dojem	98,0%

Tab. 2 - Úspešnosť jednotlivých položiek v dotazníku

8 Záver

Po zoznámení sa s rôznymi logickými hrami a ich pravidlami som pokračoval naštudovaním hry Futoshiki. Okrem histórie tejto hry som zistil jej pravidlá a možné postupy pri riešení hracieho poľa. Nasledovali mnohé pokusy riešenia tejto hry pre lepšie zoznámenie sa s princípmi a logikou hry. Až potom mohli nasledovať návrhy jednotlivých algoritmov nutných pre tvorbu aplikácie.

Na začiatku som navrhol dve algoritmy - pre riešenie hry a pre tvorbu zadania. Neskôr vznikol tretí algoritmus ako odpoveď na potrebu kontrolovať užívateľské riešenie. Dôvodom bolo to, že hra nemusí mať len jedno možné riešenie a tak nie je možné porovnávať vyplnené číslo s číslom na rovnakej pozícii v generátore. Návrhy boli detailné no nie úplné a tak sa pri implementácii niektorých častí algoritmov vyskytli isté zmeny.

Samotná implementácia algoritmov bola jednoduchá kvôli rozvrhnutiu tried a metód. Hlavnou myšlienkou bola modularita. Vytvorené boli štyri triedy - *Main*, *Generator*, *Solver*, *Cell*. Hlavná trieda volá triedy obsluhujúce generátor a riešiteľa. Tieto vytvárajú vlastné dvojrozmerné polia buniek. Obsah týchto polí sa následne skopíruje do poľa v hlavnej triede. Toto pole je vykreslené v paneli aplikácie.

Vykresľovanie je súčasťou užívateľského rozhrania. Okrem spomínaného panelu obsahuje okno aplikácie i ďalšie súčasti. Tými sú tlačidlá s priradenými úlohami a ponuky na zmenu nastavení. Uplatnil som postupy pre tvorbu užívateľských rozhraní. Preto tvorba tohto rozhrania bola časovo náročným no nevyhnutným procesom skladajúcim sa z opakovaných testov programu. Poznatky z každého testu som uplatnil v nasledujúcej verzii aplikácie. Takto bolo možné odchytiť chyby v programe a tiež upraviť vzhľad či správanie aplikácie. Najvýraznejšou úpravou bol spôsob zobrazovania okna s výberom (viď obrázok 15). V predchádzajúcej verzii bolo možné do bunky vkladať i hodnoty, ktoré sa v danom riadku či stĺpci už nachádzajú. Pre výrazný užívateľský záujem bola zmenená logika tohto výberu. Nová verzia to neumožňuje a tieto hodnoty sú vyznačené ako nevkladateľné (ich tlačidlá sú neaktívne). Špeciálnym prvkom programu je ukladanie nastavení pri zmene a ich načítavanie pri ďalšom spustení hry Futoshiki.

Na záver som vytvoril dotazník a ten som predložil vzorke ľudí, ktorej som tiež poskytol aplikáciu na zhodnotenie. Z výsledkov som vytvoril štatistiku referujúcu o úspešnosti každej z častí no i jednotlivých celkov. Porovnal som celkovú úspešnosť aplikácie s úspešnosťou, ktorú som si na začiatku zvolil ako požadovanú. Zistením bolo, že aplikácia už dostatočne vyhovuje kritériám a tak je možné ukončiť tvorbu programu.

Literatúra

[1] Futoshiki [cit. 2010-01-22]. Dostupné na URL: < http://en.wikipedia.org/wiki/Futoshiki>

[2] Sudoku [cit. 2010-01-22]. Dostupné na URL: < http://en.wikipedia.org/wiki/Sudoku>

[3] Masyu [cit. 2010-05-04]. Dostupné na URL: < http://en.wikipedia.org/wiki/Masyu>

[4] Puzzler Media, Puzzler Futoshiki, Carlton Books, 2007, ISBN 1844426106

[5] Joel Spolsky, User Interface Design for Programmers, Apress, ISBN 1-893115-94-1

[6] Jenny Le Peuple, Robert Scane: User Interface Design, Crucial, ISBN 1-903337-19-4

[7] Sharon Zakhour a kol., Java 6 Výukový kurz, Computer Press, 2007, ISBN 978-80-251-1575-6

[8] David M. Geary, Graphic Java 2 - Volume 2: Swing, Prentice Hall, ISBN 0-13-079667-0

[9] John Zukowski, The Definitive Guide to Java Swing - Third Edition, Apress, ISBN 1-590-59447-9

[10] dom4j.jar 1.6.1 [cit. 2010-05-04]. Dostupné na URL: < http://www.dom4j.org >

[11] jaxen.jar 1.1.2 [cit. 2010-05-11]. Dostupné na URL: < http://jaxen.codehaus.org >

Zoznam príloh

Príloha 1. Dotazník k aplikácii Futoshiki
Príloha 2. Metriky programu
Príloha 3. Pomocník k aplikácii
Príloha 4. CD (zdrojový kód, spustiteľná verzia aplikácie, technická správa, programová dokumentácia, pomocník k hre vo viacerých jazykoch)

Príloha 1 - Dotazník k aplikácii Futoshiki

Prosím zvoľte LEN JEDNU odpoveď pri každej položke. Zvolenú možnosť označte HRUBÝM PÍSMOM. V prípade osobného komentára k aplikácii použite priestor pre pripomienky a návrhy.

Nevyhovujúce - 0% S výhradami - 25% Vyhovujúce - 50% Dobré - 75% Vynikajúce - 100%

Ako by ste zhodnotili VZHĽAD nasledujúcich častí :					
Tlačidlá	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúce
Okno hry	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúce
Hracie pole	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúce
Nastavenia	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúce
Pozadie okna	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúce
Farby	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúce

Ako by ste zhodnotili FORMU/UMIESTNENIE nasledujúcich častí :					
Tlačidlá	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúce
Okno hry	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúce
Hracie pole	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúce
Nastavenia	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúce
Pozadie okna	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúce
Farby	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúce

Ako by ste zhodnotili OVLÁDATEĽNOSŤ nasledujúcich častí :					
Vyriešenie hry	Nevyhovujúca	S výhradami	Vyhovujúca	Dobrá	Vynikajúca
Kontrola poľa	Nevyhovujúca	S výhradami	Vyhovujúca	Dobrá	Vynikajúca
Vlastné pole	Nevyhovujúca	S výhradami	Vyhovujúca	Dobrá	Vynikajúca
Výber čísla/znaku	Nevyhovujúca	S výhradami	Vyhovujúca	Dobrá	Vynikajúca
Nastavenia	Nevyhovujúca	S výhradami	Vyhovujúca	Dobrá	Vynikajúca

Ako by ste CELKOVO zhodnotili nasledujúce časti :					
Hracie pole	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúce
Štýly	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúce
Ponuky/Tlačidlá	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúce
Zvuky	Nevyhovujúce	S výhradami	Vyhovujúce	Dobré	Vynikajúca

Pripomienky/Návrhy :

Príloha 2 - Metriky programu

- **Počet zdrojových súborov** - 5 (4 súbory typu '*.java' + 1 súbor typu '*.form')[1]
- **Veľkosť zdrojových súborov** - 229 723 bajtov
- **Počet externých súborov využívaných aplikáciou** - 17 (3 obrázkové súbory, 2 knižnice, 11 zvukových súborov, 1 dokument)
- **Veľkosť externých súborov** - 2 478 296 bajtov
- **Veľkosť spustiteľnej aplikácie** - 136 622 bajtov

Počet vytvorených metód[2]

Trieda	Počet
Main.java	20
Generator.java	9
Solver.java	8
Cell.java	22
Spolu	**59**

Počet riadkov zdrojového textu

Trieda	Počet
Main.java	2729
Generator.java	330
Solver.java	265
Cell.java	248
Spolu	**3572**

Počet testovacích spustení a doba generovania/vyriešenia poľa[3]

Náročnosť	Počet spustení	Veľkosť	Priemerná doba generovania [s]	Priemerná doba vyriešenia [s]
Ľahká	40	5x5	0,91	0,97
	40	7x7	0,98	0,98
	40	9x9	1,09	1,00
Stedná	40	5x5	0,91	0,99
	40	7x7	0,98	1,04
	40	9x9	1,09	1,11
Ťažká	40	5x5	0,91	1,03
	40	7x7	0,98	1,11
	40	9x9	1,09	1,21
Spolu	**360**	***Priemer [s]***	**0,99**	**1,05**

1.) typ '*.form' je využitý pre zmenu generovaného kódu
2.) bez konštruktorov a metód generovaných nástrojom Swing
3.) vlastné testy, nie užívateľské

Príloha 3 - Pomocník k aplikácii

Hlavné (Futoshiki) okno

Toto okno obsahuje hracie pole a akčné elementy, ktoré buď menia nastavenia alebo vykonávajú interakciu (kontrola, riešenie, resetovanie) s hracím poľom.

Tlačidlá

Je 5 dostupných tlačidiel. Každé má špecifickú funkciu, ktorá vykonáva interakciu s hracím poľom a/alebo otvára informačné okno.

- Kontrola Riešenia
Kontroluje či súčasné pole je správne vyriešenou hrou Futoshiki alebo nie. V oboch prípadoch sa objaví okno informujúce o výsledku kontroly.

- Resetuj Pole
Nastaví hracie pole na jeho úvodné nastavenie. Všetky bunky vyplnené užívateľom alebo aplikáciou sú vyprázdnené.

- Vyrieš Hru
Vyrieši hru za užívateľa. Nahrádza hodnoty buniek vyplnených užívateľom vlastnými hodnotami, ktoré aplikácia vypočítala.

- Nová Hra
Vygeneruje nové hracie pole.

- Vlastná Hra
Vyprázdni celé pole a umožní užívateľovi vyplniť všetky bunky, dokonca aj medzery medzi bunkami.

Ponuky

Výber ponúk je vo vrchnej časti okna. Obsahujú nastavenia aplikácie, pravidlá a pomocníka. Ak je vybraná položka niektorej ponuky nastavenia sa uložia.

- Veľkosť
Zmení veľkosť hracieho poľa. Pri zmene okamžite generuje nové pole.

- Náročnosť
Náročnosť hry sa mení podľa zvolenej možnosti. Taktiež okamžite generuje nové pole.

- Vzhľad
Farby elementov okna ako aj obrázok na pozadí sa zmení podľa zvoleného vzhľadu.

- Jazyk
Všetky nápisy sa preložia do zvoleného jazyka.

- Ďalšie
Obsahuje element na zapnutie/vypnutie zvuku. Pravidlá hry Futoshiki sa zobrazia v novom okne keď je daná položka ponuky zvolená. Tiež sa tu nachádza odkaz na pomocníka.

Hracie pole

Toto je hlavná (a najväčšia) časť okna. Hracie pole je tvorené bunkami. Sú dva typy – viditeľné a neviditeľné. Viditeľné bunky obsahujú čísla. Ak je bunka ‘označiteľná’ otvorí sa nové okno po kliknutí na takúto bunku. Toto okno obsahuje čísla na výber. Po výbere čísla a kliknutí na neho je jeho hodnota vložená do vyznačenej (ružovej) bunky. Neviditeľné bunky sú ‘označiteľné’ len pri tvorbe vlastného poľa. Výber tu obsahuje štyri symboly ‘väčší ako’/’menší ako’. Keď je nejaký symbol zvolený, vloží sa do vyznačenej (ružovej) bunky. Obe vyskakujúce okná tiež obsahujú prázdne tlačidlo, ktoré môže zmazať vyplnenú hodnotu.